ASP.NET Core MVC Introdução

Programação Visual

Sumário

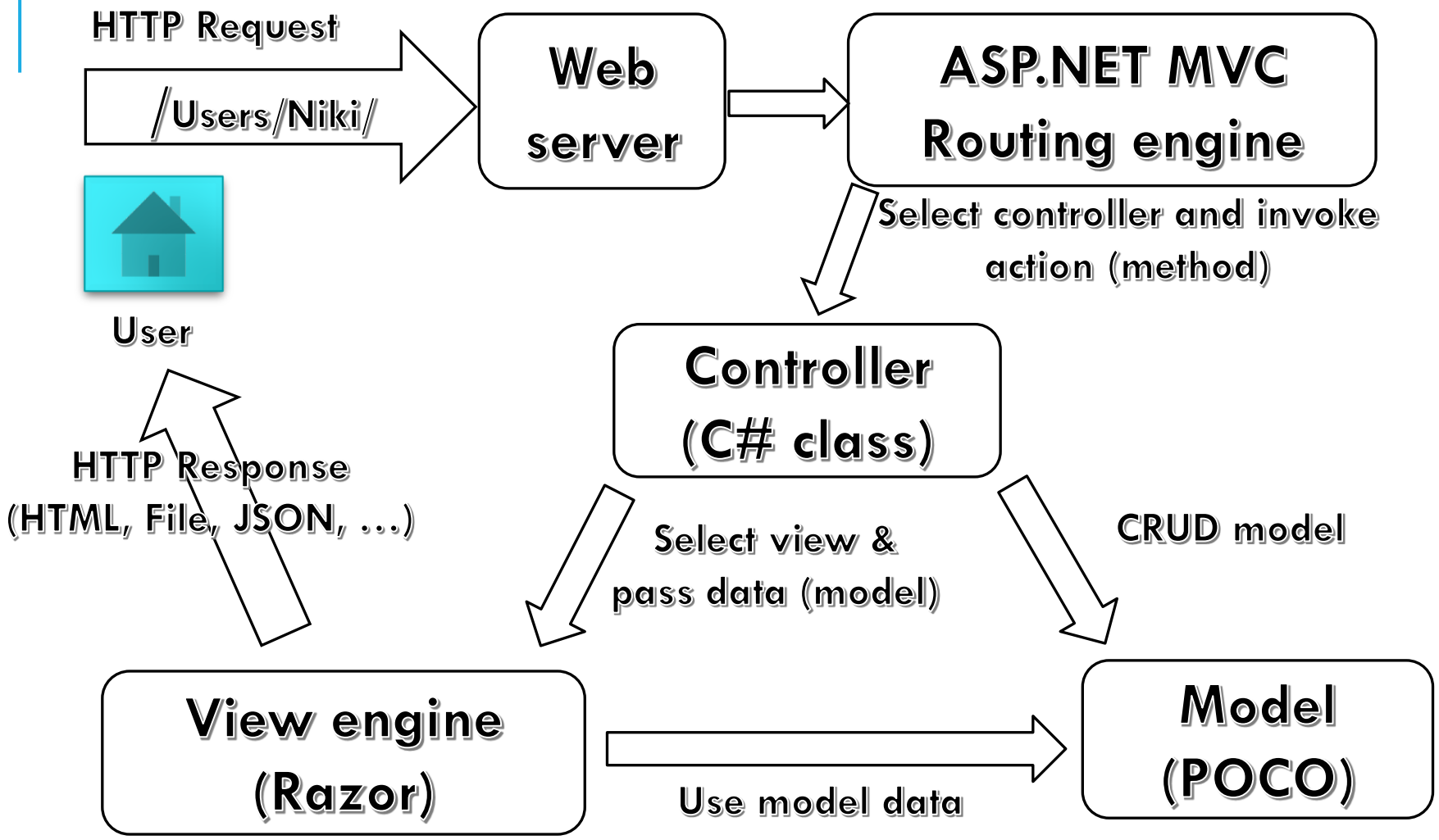
- O que é ASP.NET Core?
- Ciclo de vida de um pedido
- Encaminhamento
- Controladores e acões
- Vistas
- Linguagem Razor
- Layout

O que é a ASP.NET Core?

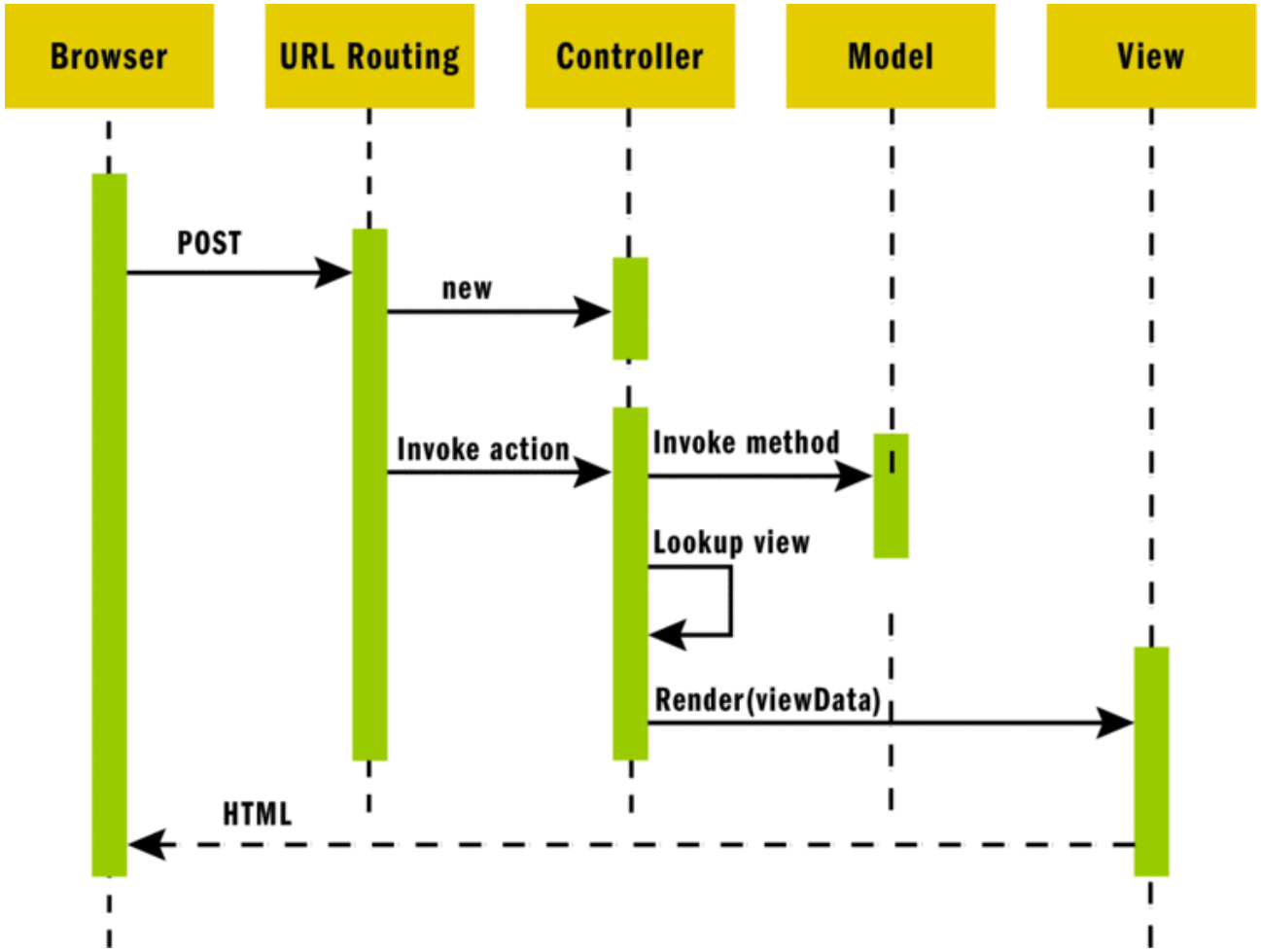
ASP.NET Core é uma *framework* multiplataforma, *open source* de alto rendimento para a construção de aplicações modernas, ligadas pela internet e com a capacidade utilização de computação na nuvem.

Tipo de Aplicação	Tipo de Desenvolvimento	Observações
MVC	Desenvolvimento WEB UI server-side	Padrão MVC
Razor Pages	Desenvolvimento WEB UI server-side	Single Page
Blazor	Desenvolvimento WEB UI cliente-side	SPA em WebAssembly
WEB API	Desenvolvimento de serviços RESTful	Serviços "MC"
SignalR	Desenvolvimento Real-time com comunicação bidirecional	Real-time
gRPC	Desenvolvimento de serviços com base em RPC	RPC

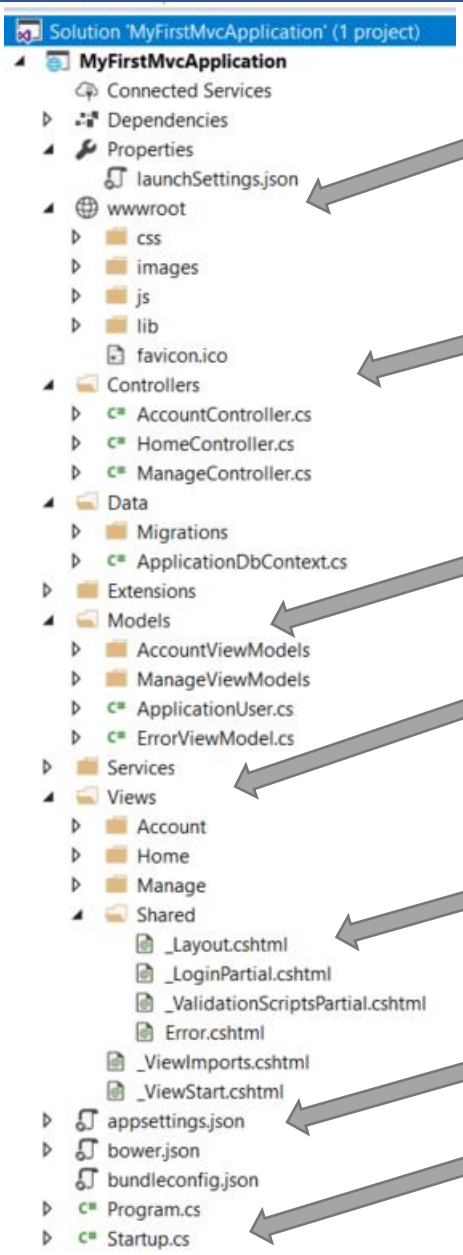
Ciclo de vida de um pedido



Ciclo de vida de um pedido



MSVS: ASP.NET Core Web Project + MVC



Ficheiros estáticos (CSS, Images, javascript, etc.)

Controladores e ações

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        ViewBag.Message = "Index page.";
        return View();
    }
}
```

Modelos

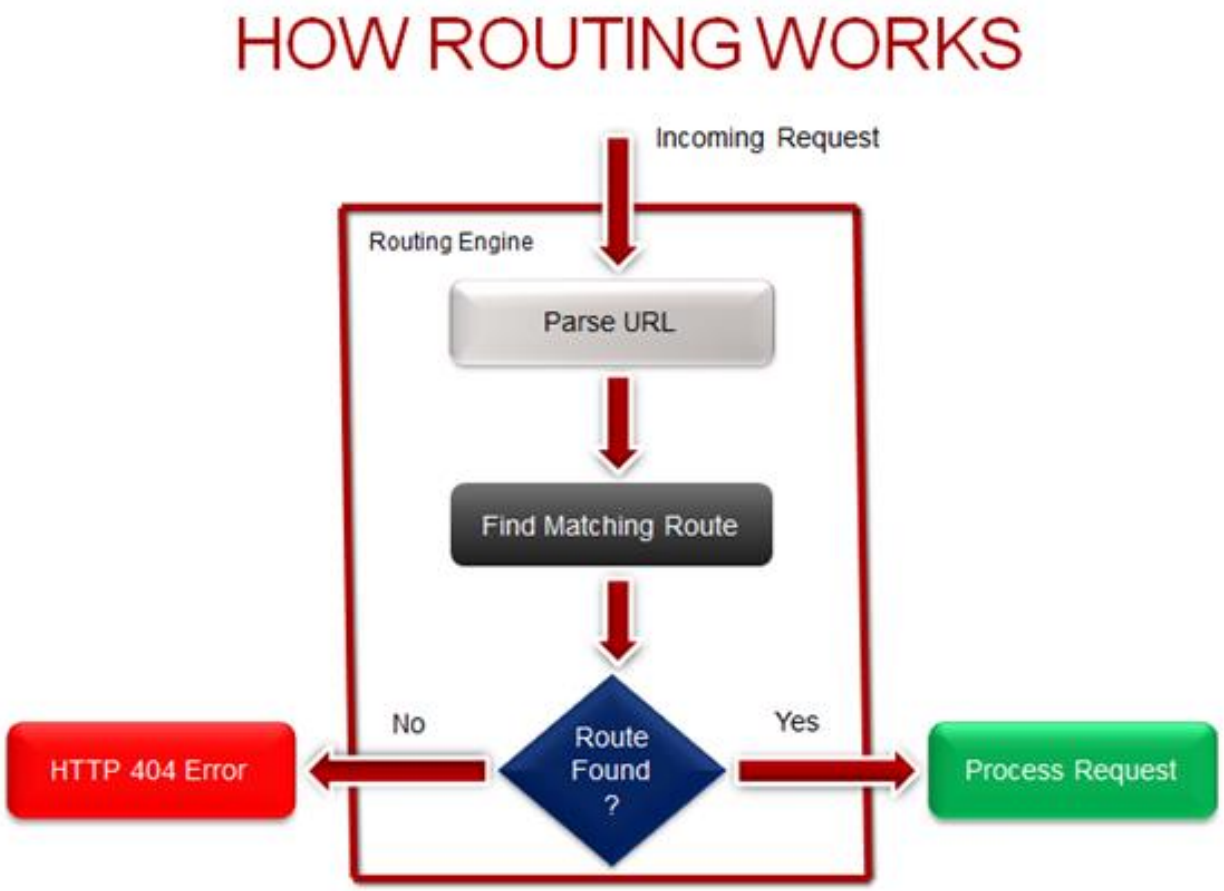
View templates

_Layout.cshtml – master page (main template)

appsettings – ficheiro de configuração

Startup – ponto de entrada da aplicação

Encaminhamento

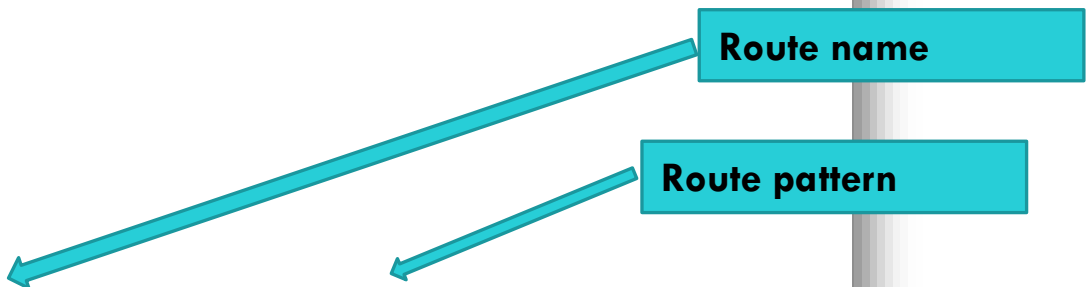


Encaminhamento

- Mapeamento entre padrões, uma combinação de controlador + ação + parâmetros
- Algoritmo
  - O primeiro *match* ganha
- As regras de encaminhamento são definidas dentro do método **`Configure()`** da classe **`Startup`**

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    ...

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}
```



Controladores

O elemento central do padrão MVC;

Devem estar sempre numa pasta com o nome Controllers;

Por convenção os nomes dos controladores devem ter o sufixo Controller: "`NameController`";

Os Routers instanciam um controlador por cada pedido:

- Todos os pedidos são *mapeados* para uma ação específica.

Todos os controladores herdam da classe Controller

- Fornece o acesso ao `Request` (informação do pedido)

Ações

Ações representam o destino final dos pedidos:

- São métodos públicos do controlador;
- Não estáticos;
- Sem restrições ao valor de retorno.

As ações retornam normalmente um **IActionResult**:

```
public IActionResult Contact()
{
    ViewData["Message"] = "Your contact page.";

    return View();
}
```

MVC – Resultados das Ações

- Respondem a um pedido do Browser;
- Implementam a interface `IActionResult`;
- Vários tipos de retorno:

Name	Framework Behavior	Producing Method
`ContentResult`	Returns a string literal by default, may return other types.	Content
`EmptyResult`	No response. For command operations like delete, update, create	

`FileContentResult`
`PhysicalFileResult`
`FileStreamResult`
`VirtualFileResult`
	Return the contents of a file. Derived from `FileResult`	File

MVC – Resultados das Ações

Name	Framework Behavior	Producing Method
`ChallengeResult`	Authentication credential not valid or not present. Returns an HTTP 401 status code	
`UnauthorizedResult`	Returns an HTTP 401 status code and nothing else	

`RedirectResult`
`RedirectToActionResult`
`RedirectToRouteResult`
`LocalRedirectResult`
	Redirects the client to a new URL. Derived from `RedirectToActionResult`	Redirect / RedirectPermanent

MVC – Resultados das Ações

Name	Framework Behavior	Producing Method
`ForbidResult`	To refuse a request to a particular resource. Returns an HTTP 403 status	
`JsonResult`	Returns data in JSON format	Json
`ViewResult` `PartialViewResult`	Response is the responsibility of a view engine- To render a view or part of it.	View / PartialView
`ViewComponentResult`	Returns html content for a view	
`SignInResult` `SignOutResult`	Sign in or sign out the user based on provided mechanism	

… → Existem mais tipos de retorno:

MVC – Resultados das Ações

- ASP.NET MVC mapeia a informação do pedido HTTP para os parâmetros das ações de diferentes formas:
  - **Routing engine** fornece os parâmetros das ações:
    - http://localhost/Users/NikolayIT
    - Routing pattern: Users/{username}
  - **URL query string** fornece os parâmetros:
    - /Users/ByUsername?username=NikolayIT
  - **HTTP post** pode igualmente fornecer os parâmetros:

```
public IActionResult ByUsername(string username)
{
    return Content(username);
}
```

MVC – Vistas

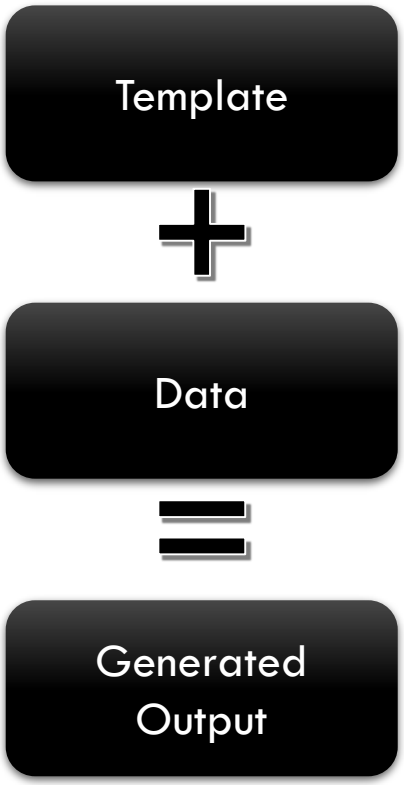
- Templates HTML da aplicação;
- Usa o **Razor** view engine:
  - Executa o código e fornece o HTML;
  - Disponibiliza vários *html ou ag helpers* para a geração do HTML;
- É possível passar informação para as vistas através de: `ViewData`, `ViewBag` e `Model` (strongly-typed views);
- As Views suportam ***master pages*** (layout views);
- Outras vistas podem ser renderizadas (partial views).

MVC – Linguagem Razor (Vistas)

- Template markup syntax;
- Simple-syntax view engine;
- Baseado na linguagem C#
- Permite ao programador usar *normalmente* a linguagem HTML;
- É uma abordagem focada no código fornecido como *template*, com uma transição *suave* entre HTML e código:
  - Na sintaxe Razor os blocos de código começam com o caracter @ e não necessitam que o bloco tenha um fecho explicito.

MVC – Linguagem Razor (Vistas)

- Compacto, Expressivo, e Fluido:
  - Suficientemente inteligente para distinguir o HTML do código.
- Fácil de aprender;
- Não é uma nova linguagem;
- Funciona com qualquer editor de texto;
- Tem um excelente *Intellisense*:
  - Built in no Visual Studio.
- Facilmente testável com Unit:
  - Sem necessitar de um controlador ou de um WebService.



MVC – Linguagem Razor (Vistas)

- **@** – Para valores (HTML encoded)

```
<p>
    Current time is: @DateTime.Now!!!
    Not HTML encoded value: @Html.Raw(someVar)
</p>
```

- **@{ … }** – Para blocos de código (Manter a vista simples!)

```
@{
    var productName = "Energy drink";
    if (Model != null)
    {
        productName = Model.ProductName;
    }
    else if (ViewBag.ProductName != null)
    {
        productName = ViewBag.ProductName;
    }
}
<p>Product "@productName" has been added in your shopping cart</p>
```

MVC – Linguagem Razor (Vistas)

- If, else, for, foreach, etc.
  - As linhas de *markup* HTML podem ser incluídas em qualquer parte
  - @: – Para a renderização de texto simples

```
<div class="products-list">
@if (Model.Products.Count() == 0)
{
     <p>Sorry, no products found!</p>
}
else
{
     @:List of the products found:
     foreach(var product in Model.Products)
     {
          <b>@product.Name, </b>
     }
}
</div>
```

MVC – Linguagem Razor (Vistas)

- Comentários

```
@*
    A Razor Comment
*@
@{
    //A C# comment

    /* A Multi
         line C# comment
    */
}
```

- E em relação a "@" e emails?

```
<p>
    This is the sign that separates email names from domains:
@@<br />
    And this is how smart Razor is: spam_me@gmail.com
</p>
```

MVC – Linguagem Razor (Vistas)

- **@(…)** – Expressão de código explícita

```
<p>
    Current rating(0-10): @Model.Rating / 10.0      @* 6 / 10.0 *@
    Current rating(0-1): @(Model.Rating / 10.0)     @* 0.6 *@
    spam_me@Model.Rating                                @* spam_me@Model.Rating *@
    spam_me@(Model.Rating)                              @* spam_me6 *@
</p>
```

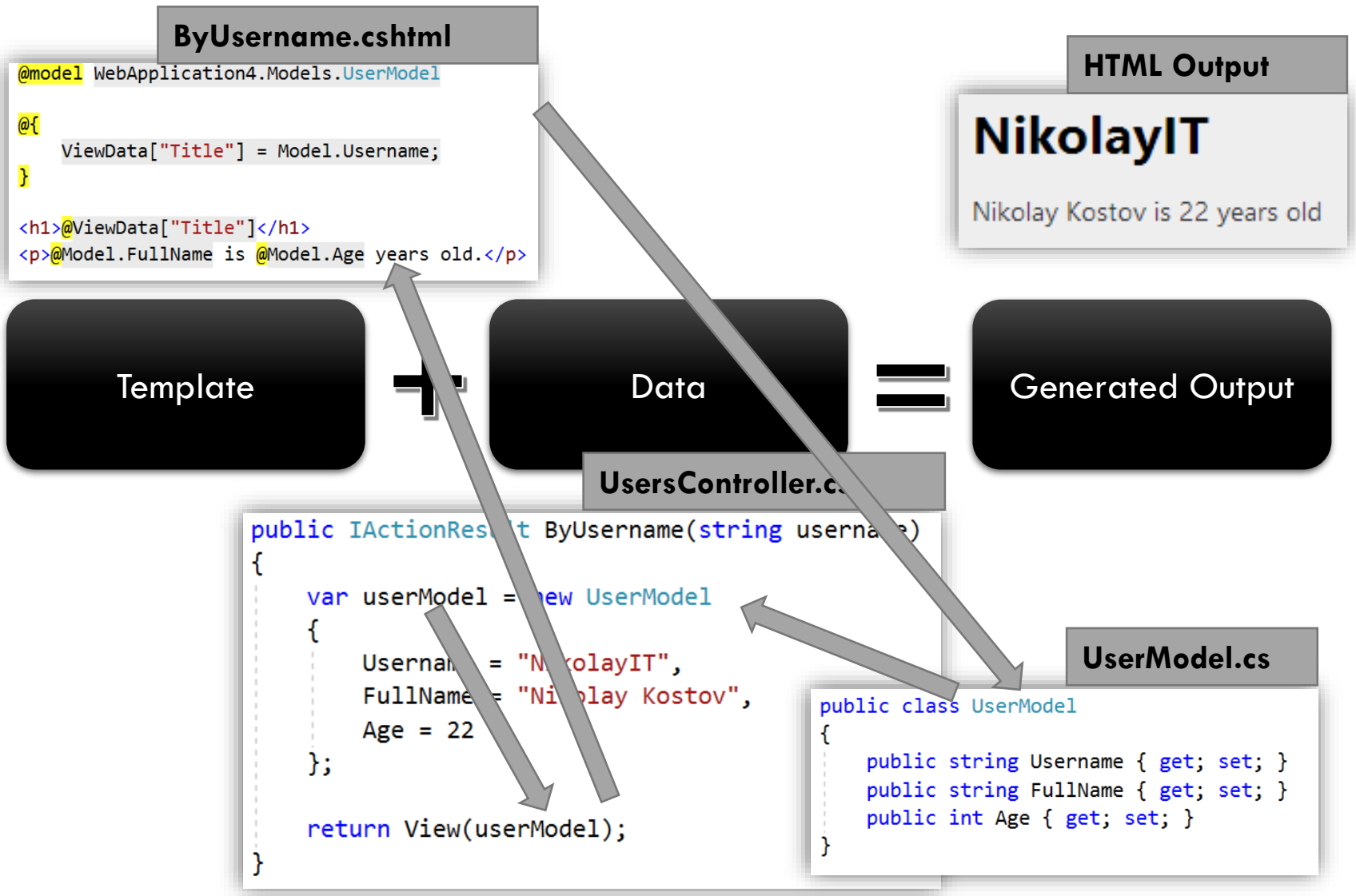
- **@using** – para incluir o *namespace* na vista
- **@model** – para definir o modelo a ser usado na vista

```
@using MyFirstMvcApplication.Models;
@model UserModel
<p>@Model.Username</p>
```

MVC – Passar dados para as vistas

- Vistas Strongly-typed:
  - Action: `return View(model);`
  - View: `@model ModelDataType`.
- Através de ViewData (dictionary)
  - `ViewData["message"] = "Hello World!";`
  - View: `@ViewData["message"]`
- Através de ViewBag (dynamic type):
  - Action: `ViewBag.Message = "Hello World!";`
  - View: `@ViewBag.Message`.

MVC – Passar dados para as vistas



MVC – Layout

- Define um modelo comum para o *site;*
- Similar às ASP.NET *master pages* (mas melhor!);
- Razor view engine renderiza o conteúdo de dentro para fora;
  - Primeira a vista, depois o Layout
- **`@RenderBody()`** – indica o local para o "preenchimento" que as vistas baseadas neste *layout* têm de preencher fornecendo o conteúdo.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
    </head>
    <body>
        <nav>@* Menu *@</nav>
        <div id="body">
            @RenderBody()
        </div>
        <footer>@* Footer *@</footer>
    </body>
</html>
```

MVC – Vistas e Layout

- As vista não necessitam de especificar o *layout* uma vez que o valor por omissão é dado em _ViewStart.cshtml:
  - `~/Views/_ViewStart.cshtml` (Código para todas as vistas)

- Cada vista pode especificar páginas de layout particulares

```
@{
      Layout = "~/Views/Shared/_UncommonLayout.cshtml";
}

@{
      Layout = null;
}
```

Referências

- Telerik Software Academy
  - academy.telerik.com

- https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0